

The Embedded Design Handbook complements the primary documentation for the Altera® tools for embedded system development. It describes how to most effectively use the tools, and recommends design styles and practices for developing, debugging, and optimizing embedded systems using Altera-provided tools. The handbook introduces concepts to new users of Altera's embedded solutions, and helps to increase the design efficiency of the experienced user.

This section includes the following chapters:

- [Chapter 1, First Time Designer's Guide](#)



For information about the revision history for chapters in this section, refer to each individual chapter for that chapter's revision history.



## Introduction

Altera® provides various tools for development of hardware and software for embedded systems. This handbook complements the primary documentation for these tools by describing how to most effectively use the tools. It recommends design styles and practices for developing, debugging, and optimizing embedded systems using Altera-provided tools. The handbook introduces concepts to new users of Altera's embedded solutions, and helps to increase the design efficiency of the experienced user.

This handbook is not a comprehensive reference guide. For general reference and detailed information, refer to the primary documentation cited in this handbook.

This first chapter of the handbook contains information about the Altera embedded development process and procedures for the first time user. The remaining chapters focus on specific aspects of embedded development for Altera FPGAs.

### First Time Designer's Guide Introduction

This chapter is for first time users of Altera's embedded development tools for hardware and software development. The chapter provides information about the design flow and development tools interaction, and describes the differences between the Nios® II processor flow and a typical discrete microcontroller design flow.

However, this chapter does not replace the basic reference material for the first time designer, such as the *Nios II Processor Reference Handbook*, the *Nios II Software Developer's Handbook*, volumes 4 and 5 of the *Quartus II Handbook*, and the *Nios II Flash Programmer's Guide*.

### FPGAs and Soft-Core Processors

FPGAs can implement logic that functions as a complete microprocessor while providing many flexibility options.

An important difference between discrete microprocessors and FPGAs is that an FPGA contains no logic when it powers up. Before you run software on a Nios II based system, you must configure the FPGA with a hardware design that contains a Nios II processor. To configure an FPGA is to electronically program the FPGA with a specific logic design. The Nios II processor is a true soft-core processor: it can be placed anywhere on the FPGA, depending on the other requirements of the design. Three different sizes of the processor are available, each with flexible features.

To enable your FPGA-based embedded system to behave as a discrete microprocessor-based system, your system should include the following:

- A JTAG interface to support FPGA configuration and hardware and software debugging
- A power-up FPGA configuration mechanism

If your system has these capabilities, you can begin refining your design from a pretested hardware design loaded in the FPGA. Using an FPGA also allows you to modify your design quickly to address problems or to add new functionality. You can test these new hardware designs easily by reconfiguring the FPGA using your system's JTAG interface.

The JTAG interface supports hardware and software development. You can perform the following tasks using the JTAG interface:

- Configure the FPGA
- Download and debug software
- Communicate with the FPGA through a UART-like interface (JTAG UART)
- Debug hardware (with the SignalTap® II embedded logic analyzer)
- Program flash memory

After you configure the FPGA with your Nios II processor-based design, the software development flow is similar to the flow for discrete microcontroller designs.

## Embedded System Design

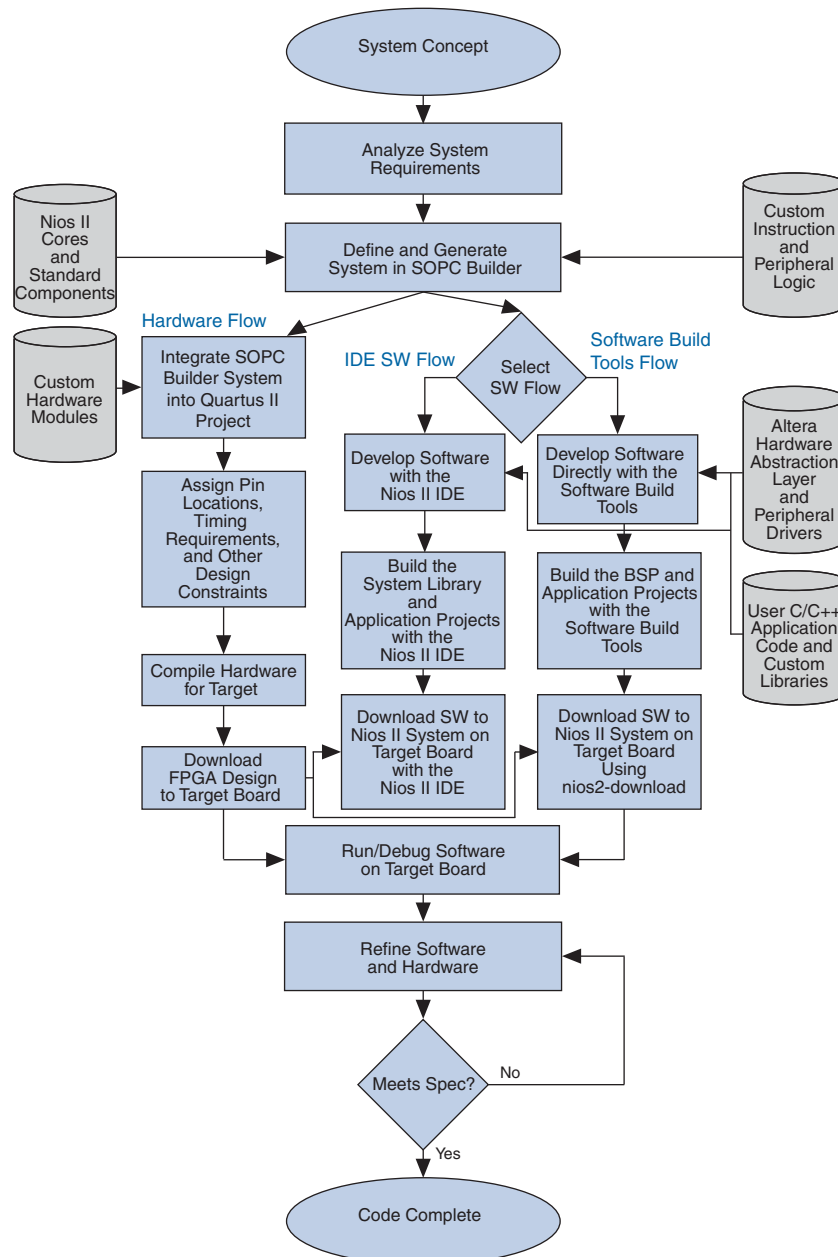
### FPGA Hardware Design

Whether you are a hardware designer or a software designer, read the *Nios II Hardware Development Tutorial* to start learning about designing embedded systems on an Altera FPGA. The “Nios II System Development Flow” section is particularly useful in helping you to decide how to approach system design using Altera's embedded hardware and software development tools. Altera recommends that you read this tutorial before starting your first design project. The tutorial teaches you the basic hardware and software flow for developing Nios II processor-based systems.

Designing with FPGAs gives you the flexibility to implement some functionality in discrete system components, some in software, and some in FPGA-based hardware. This flexibility makes the design process more complex. The SOPC Builder system design tool helps to manage this complexity. Even if you decide a soft-core processor doesn't meet your application's needs, SOPC Builder can still play a vital role in your system by providing mechanisms for peripheral expansion or processor off load.

Figure 1-1 illustrates the FPGA hardware design process and Nios II software flow.

Figure 1-1. System Design Flow



Although you develop your FPGA-based design in SOPC Builder, you must perform the following tasks in other tools:

- Connect signals from your FPGA-based design to your board level design
- Connect signals from your SOPC Builder system to other signals in the FPGA logic
- Constrain your design

## Connecting Your FPGA Design to Your Board

To connect your FPGA-based design to your board-level design, perform the following two tasks:

1. Identify the top level of your FPGA design.
2. Assign signals in the top level of your FPGA design to pins on your FPGA using any of the methods mentioned at the Altera I/O Management, Board Development Support, and Signal Integrity Analysis Resource Center, at [www.altera.com/support/software/io-board/sof-qts-io.html](http://www.altera.com/support/software/io-board/sof-qts-io.html)



The top level of your FPGA-based design might be your SOPC Builder system. However, the FPGA can include additional design logic.

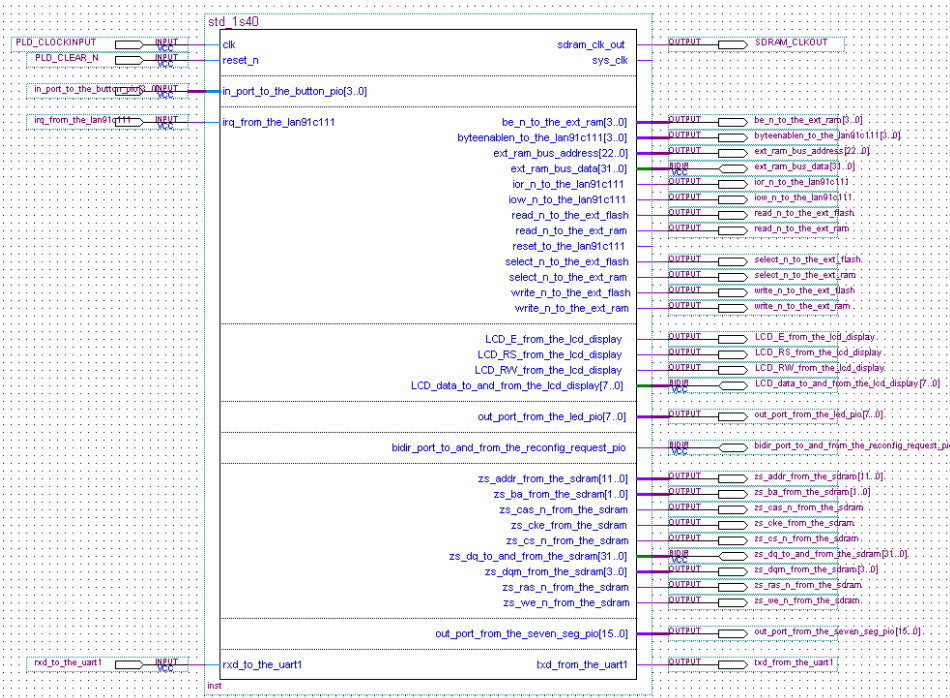
## Connecting Signals to your SOPC Builder System

You must define the clock and reset pins for your SOPC Builder system. You must also define each I/O signal that is required for proper system operation. [Figure 1-2](#) shows the top level block diagram of an SOPC Builder system that includes a Nios II processor. The large symbol in this top-level diagram, labeled **std\_1s40**, represents the SOPC Builder system. The flag-shaped pin symbols in this diagram represent off-chip (off-FPGA) connections.




For more information about connecting your FPGA pins, refer to the Altera I/O Management, Board Development Support, and Signal Integrity Analysis Resource Center web page.

**Figure 1-2.** Top Level Block Diagram



## Constraining Your FPGA-Based Design

To ensure your design meets timing and other requirements, you must constrain the design to meet these requirements explicitly using tools provided in the Quartus® II software or by a third party EDA provider. The Quartus II software uses your constraint information during design compilation to achieve Altera's best possible results.

-  Altera's third-party EDA partners and the tools they provide are listed at [www.altera.com/products/software/partners/eda\\_partners/eda-tools.html](http://www.altera.com/products/software/partners/eda_partners/eda-tools.html)


## SOPC Builder Design

SOPC Builder simplifies the task of building complex hardware systems on an FPGA. SOPC Builder allows you to describe the topology of your system using a graphical user interface (GUI) and then generate the hardware description language (HDL) files for that system. The Quartus II software compiles the HDL files to create an FPGA programming file.

-  For additional information about SOPC Builder, refer to *Volume 4: SOPC Builder* of the *Quartus II Handbook*.


SOPC Builder allows you to choose the processor core type and the level of cache, debugging, and custom functionality for each Nios II processor. Your design can use on-chip resources such as memory, PLLs, DSP functions, and high-speed transceivers. You can construct the optimal processor for your design using SOPC Builder.


After you construct your system using SOPC Builder, and after you add any required custom logic to complete your top-level design, you must create pin assignments using the Quartus II software. The FPGA's external pins have flexible functionality, and a range of pins is available to connect to clocks, control signals, and I/O signals.

-  For information about how to create pin assignments, refer to the Quartus II online Help and to the *I/O Management* chapter in volume 2 of the *Quartus II Handbook*.

Altera recommends that you start your design from a small pretested project and build it incrementally. Start with one of the many SOPC Builder example designs provided with the Nios II Embedded Design Suite (EDS), or with a design example from the *Nios II Hardware Development Tutorial*.


The Nios II EDS includes several SOPC Builder-based hardware example designs and corresponding software examples. The software examples are located in the hardware project directory of your Altera Nios development board type—for example, `$$SOPC_KIT_NIOS2\examples\verilog\niosII_cycloneII_2c35`—in the `software_examples` subdirectory for your design type.

-  For more information about the examples provided in the Nios II EDS, refer to the "Using Nios II Example Design Scripts" section of the *Using the Nios II Software Build Tools* chapter of the *Nios II Software Developer's Handbook*.

-  As you add each hardware component to the system, test it with software. If you do not know how to develop software to test new hardware components, Altera recommends that you work with a software engineer to test the components.

After you run a simple software design—such as the simplest built-in example, Hello World Small—build individual systems based on this design to test the additional interfaces or custom options that your system requires. Altera recommends that you start with a simple system that includes a processor with a JTAG debug module, an on-chip memory component, and a JTAG UART component, and create a new system for each new untested component, rather than adding in new untested components incrementally.

After you verify that each new hardware component functions correctly in its own separate system, you can combine the new components incrementally in a single SOPC Builder system. SOPC Builder supports this design methodology well, by allowing you to add components and regenerate the project easily.

 For detailed information about how to implement the recommended incremental design process, refer to the *Verification and Board Bring-Up* chapter of the *Embedded Design Handbook*.

### Design Replication


The recommended design flow requires that you maintain several small SOPC Builder systems, each with its Quartus II project and the software you use to test the new hardware. An SOPC Builder design requires the following files and folders:


- Quartus II project file (**.qpf**)
- Quartus II settings file (**.qsf**)

The **.qsf** file contains all of the device, pin, timing, and compilation settings for the Quartus II project.

- A top level design file – schematic (**.bdf**), Verilog HDL (**.v**), or VHDL (**.vhd**)

If SOPC Builder generates your top-level design file, you do not need to preserve a separate top-level file.

 SOPC Builder generates most of the HDL files for your system, so you do not need to maintain them when preserving a project. You need only preserve the HDL files that you add to the design directly.


 For details about the design file types, refer to the Quartus II online Help.

- Internal SOPC Builder description file (**.sopc**)
- SOPC Builder description file (**.sopcinfo**)

This file contains an XML description of your SOPC Builder system. SOPC Builder and downstream tools, including the software build tools, derive information about your system from this file.

- Your software application source files

To replicate an entire project (both hardware and software), first copy the required files to a separate directory, and then open the new project. You can open the new project in the Quartus II software, in SOPC Builder, or in the Nios II Integrated Development Environment (IDE). You can also create a script to automate the copying process.

 For more information about all of these files, refer to the *Archiving SOPC Builder Projects* chapter in volume 4 of the *Quartus II Handbook*.

## Customization and Acceleration

FPGA-based designs provide you with the flexibility to modify your design easily, and to experiment to determine the best balance between hardware and software implementation of your design. In a discrete microcontroller-based design process, you must determine the processor resources—cache size and built-in peripherals, for example—before you reach the final design stages. You may be forced to make these resource decisions before you know your final processor requirements. If you implement some or all of your system's critical design components in an FPGA, you can easily redesign your system as your final product needs become clear. If you use the Nios II processor, you can experiment with the correct balance of processor resources to optimize your system for your needs. SOPC Builder facilitates this flexibility, by allowing you to add and modify system components and regenerate your project easily.

Similarly, if you implement your system in an FPGA, you can experiment with the best balance of hardware and software resource usage. If you find you have a software bottleneck in some part of your application, you can consider accelerating the relevant algorithm by implementing it in hardware instead of software. SOPC Builder facilitates experimenting with the balance of software and hardware implementation. You can even design custom hardware accelerators for specific system tasks.

To help you solve system performance issues, the following acceleration methodologies are available:

- Custom peripherals
- Custom instructions
- C2H accelerated software

The method of acceleration you choose depends on the operation you wish to accelerate. To accelerate streaming operations on large amounts of data, a custom peripheral may be a good solution. Hardware interfaces (such as implementations of the Ethernet or serial peripheral interface (SPI) protocol) may also be implemented efficiently as custom peripherals. The current floating-point custom instruction is a good example of the type of operations that are typically best accelerated using custom instructions.

Working with a software or systems engineer, use the C2H Compiler to help analyze sophisticated algorithms to determine potential hardware acceleration gains. As in any hardware acceleration methodology, you must make trade-offs between performance and resource consumption. When a C compiler compiles code using a high level of optimization, the resulting executable program typically runs faster, but also often consumes more memory than similar code compiled with a lower level of optimization. Similarly, accelerators built with the C2H Compiler typically run faster than the unaccelerated code, but they consume more FPGA resources.

- For information about hardware acceleration, refer to the *Hardware Acceleration and Coprocessing* chapter of the *Embedded Design Handbook*. For information about how to use the C2H Compiler, refer to the *Nios II C2H Compiler User Guide* and to the *Optimizing Nios II C2H Compiler Results* chapter of the *Embedded Design Handbook*. For information on custom instructions, refer to the *Nios II Custom Instruction User Guide*. For information on creating custom peripherals, refer to the *Developing Components for SOPC Builder* chapter in volume 4 of the *Quartus II Handbook*.


## Software Design

This section contains brief descriptions of the software design tools provided by the Nios II EDS, the Nios II IDE development flow, and the software build tools development flow.

### Tools Description


The Nios II EDS provides the following tools for software development:

- GNU toolchain: GCC-based compiler with the GNU binary utilities

-  For an overview of these and other Altera-provided utilities, refer to the *Nios II Command-Line Tools* chapter of the *Embedded Design Handbook*.

- Nios II processor-specific port of the newlib C library
- Hardware abstraction layer (HAL)

The HAL provides a simple device driver interface for programs to communicate with the underlying hardware. It provides many useful features such as a POSIX-like application program interface (API) and a virtual-device file system.


-  For more information about the Altera HAL, refer to [The Hardware Abstraction Layer](#) section of the *Nios II Software Developer's Handbook*.

- Nios II IDE

The Nios II IDE is a GUI that supports creating, modifying, building, running, and debugging Nios II programs. It is based on the Eclipse open development platform and Eclipse C/C++ development toolkit (CDT) plug-ins.

- Nios II software build tools flow

The Nios II software build tools development flow is a scriptable, command-line based development flow that uses the software build tools independent of the Nios II IDE.

-  For more information about the Nios II software build tools flow, refer to the *Developing Nios II Software* chapter of the *Embedded Design Handbook*.

### Nios II IDE Flow

To learn about the Nios II IDE, refer to the Nios II software development tutorial. Unlike the *Nios II Hardware Development Tutorial*, this tutorial is contained in the Nios II IDE Help system. To open this Help system, in the Nios II IDE, on the Help menu, click **Welcome**. A PDF version is also available at [www.altera.com/literature/ug/ug\\_nios2\\_ide\\_help.pdf](http://www.altera.com/literature/ug/ug_nios2_ide_help.pdf).

The Nios II software development tutorial teaches you about the following key elements of the flow:

- System library project
- Software abstraction of the SOPC Builder hardware design
- Application project
- The software that drives your application

It also teaches you to develop your own software applications. However, Altera recommends that you view and begin your design with one of the available software examples that are installed with the Nios II EDS. From simple "Hello, World" programs to networking and RTOS-based software, these examples provide good reference points and starting points for your own software development projects. The Hello World Small example program illustrates how to reduce your code size without losing all of the conveniences of the HAL.



Altera recommends that you use an Altera Nios II development kit or custom prototype board for software development and debugging. Many peripheral and system-level features are available only when your software runs on an actual board.



For more detailed information, refer to the *Nios II Software Developer's Handbook*.

### Debugging Options

The Nios II EDS provides the following programs to aid in debugging your hardware and software system:

- A built-in Nios II IDE Debugger
- Several distinct interfaces to the GNU Debugger (GDB)
- A Nios II-specific implementation of the First Silicon Solutions, Inc. FS2 console (available on Windows platforms only)
- System Console, a system debug console

You can begin debugging software immediately using the built-in Nios II IDE Debugger. This debugging environment includes advanced features such as trace, watchpoints, and hardware breakpoints.

The Nios II EDS includes the following three interfaces to the GDB debugger:

- GDB console (accessible through the Nios II IDE)
- Standard GDB client (nios2-elf-gdb)
- Insight GDB interface (Tcl/Tk based GUI)

Additional GDB interfaces such as Data Display Debugger (DDD), and Curses GDB (CGDB) interface also function with the Nios II version of the GDB debugger.



For more information about these interfaces to the GDB debugger, refer to the *Nios II Command-Line Tools* and *Debugging Nios II Designs* chapters of the *Embedded Design Handbook*.

- For detailed information about the FS2 console, refer to the documentation in the `$SOPC_KIT_NIOS2\bin\fs2\doc` directory and to the *Verification and Board Bring-Up* chapter of the *Embedded Design Handbook*.

The System Console is a system debug console that provides the SOPC Builder designer with a Tcl-based, scriptable command-line interface for performing system or individual component testing. It is available in Nios II EDS version 8.0 and later.

- For detailed information about the System Console, refer to the *System Console User Guide*. On-line training is available at <http://www.altera.com/training>.

Third party debugging environments are also available from vendors such as Lauterbach Datentechnik GmbH and First Silicon Solutions, Inc.

### Command Line

You can use the Nios II IDE to create your project. The Nios II IDE guides you if you are unfamiliar with the Nios II software toolchain. It also provides easy access to newlib library functions and the HAL software layer.

However, some actions, such as rebuilding software after minor source code edits, do not require the IDE. In these cases, you may rebuild the project from a Nios II command shell, using your application's makefile. For example, to build or rebuild your software, perform the following steps:

1. Open a Nios II command shell.

To start the Nios II command shell on Windows platforms, on the Start menu, click **All Programs**. On the All Programs menu, on the Altera submenu, on the Nios II EDS *<version>* submenu, click **Nios II *<version>* Command Shell**.

On Linux platforms, type the following command:

```
$SOPC_KIT_NIOS2/sdk_shell ←
```

2. Change to the directory in which your makefile is located. If you use the Nios II IDE for development, the correct location is often the **Debug** or **Release** subdirectory of your software project directory.
3. In the command shell, type one of the following commands:  
make ←  
or  
make -s ←

**Example 1-1** illustrates the output of the make command run on a sample system.

---

### Example 1-1. Sample Output From make -s Command

---

```
[SOPC Builder]$ make -s
Creating generated_app.mk...
Creating generated_all.mk...
Creating system.h...
Creating alt_sys_init.c...
Creating generated.sh...
Creating generated.gdb...
Creating generated.x...
Compiling src1.c...
Compiling src2.c...
Compiling src3.c...
Compiling src4.c...
Compiling src5.c...
Linking project_name.elf...
```

---



If you add new files to your project or rebuild your project after significant hardware changes, you should build your project from the Nios II IDE. The Nios II IDE recreates the makefile for the new version of your system after the modifications.

### Software Build Tools Flow

The Nios II software build tools flow uses the software build tools to provide a flexible, portable, and scriptable software build environment. Altera recommends that you use this flow if you prefer a command-line environment, or if you want a set of build tools that fits easily in your preferred software or system development environment. The Nios II software build tools are the basis for Altera's future development.

The software build tools flow requires that you have an SOPC file (**.sopc**) generated by SOPC Builder for your system. The flow includes the following steps to create software for your system:

1. Create a board support package (BSP) for your system. The BSP is a layer of software that interacts with your development system. It is a makefile-based project.
2. Create your application software:
  - a. Write your code.
  - b. Generate a makefile-based project that contains your code.
3. Iterate through one or both of these steps until your design is complete.



For more information, refer to the software design examples based on this flow that are shipped with every release of the Nios II EDS. For more information about these examples, refer to the "Using Nios II Example Design Scripts" section of the *Using the Nios II Software Build Tools* chapter of the *Nios II Software Developer's Handbook*.

## Board Design Considerations

You must choose the method to configure, or program, your FPGA, and the method to boot your Nios II processor.

## Configuration

Many FPGA configuration options are available to you. The two most commonly used options configure the FPGA from flash memory. One option uses a CPLD and a CFI flash device to configure the FPGA, and the other uses a serial flash EPCS configuration device. The Nios II development kits use these two configuration options by default.

Choose the first option, which uses a CPLD and a CFI-compliant flash memory, in the following cases:

- Your FPGA is large
- You must configure multiple FPGAs
- You require a large amount of flash memory for software storage
- Your design requires multiple FPGA hardware images (safe factory images and user images) or multiple software images

EPCS configuration devices are often used to configure small, single-FPGA systems.



The default Nios II boot loader does not support multiple FPGA images in EPCS devices.



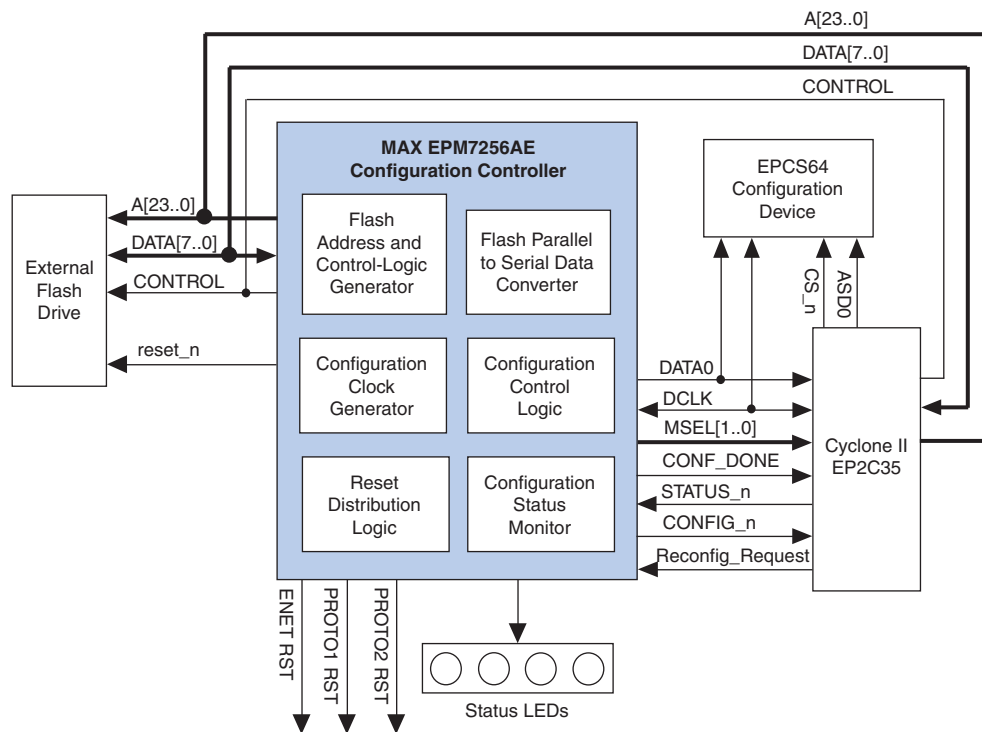
For help in configuring your particular device, refer to the device family information at [www.altera.com/products/devices/dev-index.jsp](http://www.altera.com/products/devices/dev-index.jsp).

Figure 1-3 shows the block diagram of the configuration controller used on the Nios II Development Kit, Cyclone® II Edition. This controller design is used on many of the development kits, and is a good starting point for your design.



For more information about controller designs, refer to *AN346: Using the Nios Development Board Configuration Controller Reference Designs*.

Figure 1-3. Configuration Controller for Cyclone II Devices



### altremote\_update Megafunction-Based Configuration

Newer devices such as the Cyclone III, Stratix® II, and later devices include the built-in ALTREMOTE\_UPDATE megafunction to help you configure your FPGA. For these newer devices, no additional Programmable Logic Device (PLD) is necessary for configuration control. However, older devices require a configuration controller device, as shown in Figure 1-3.


For information about the ALTREMOTE\_UPDATE megafunction, refer to the *Remote Update Circuitry Megafunction User Guide (ALTREMOTE\_UPDATE)*. The Application Selector example uses this megafunction in the Nios II Embedded Evaluation Kit (NEEK), Cyclone III Edition.

### Booting

Many Nios II booting options are available. The following options are the most commonly used:

- Boot from CFI Flash
- Boot from EPCS
- Boot from on-chip RAM

The default boot loader that is included in the Nios II EDS supports boot from CFI flash memory and from EPCS flash memory. If you use an on-chip RAM that supports initialization, such as the M4K and M9K types of RAM, you can boot from the on-chip RAM without a boot loader.

- 
- For additional information on Nios II boot methodologies, refer to [AN458: Alternative Nios II Boot Methods](#).

### Additional Design Considerations


Consider the following topics as you design your system:


- JTAG signal integrity
- Extra memory space for prototyping
- System verification

#### JTAG Signal Integrity

The JTAG signal integrity on your system is very important. You must debug your hardware and software, and program your FPGA, through the JTAG interface. Poor signal integrity on the JTAG interface can prevent you from debugging over the JTAG connection, or cause inconsistent debugger behavior.

You can use the System Console to verify the JTAG chain.


- 
- JTAG signal integrity problems are extremely difficult to diagnose. To increase the probability of avoiding these problems, and to help you diagnose them should they arise, Altera recommends that you follow the guidelines outlined in [AN428: MAX II CPLD Design Guidelines](#) and in the [Verification and Board Bring-Up](#) chapter of the *Embedded Design Handbook* when designing your board.

- 
- For more information about the System Console, refer to the [System Console User Guide](#).

#### Extra Memory Space For Prototyping

Even if your final product includes no off-chip memory, Altera recommends that your prototype board include a connection to some region of off-chip memory. This component in your system provides additional memory capacity that enables you to focus on refining code functionality without worrying about code size. Later in the design process, you can substitute a smaller memory device to store your software.

#### System Verification

- 
- For useful information about design techniques for your embedded system, refer to the [Verification and Board Bring-Up](#) chapter of the *Embedded Design Handbook*. Altera recommends that you read this chapter before you begin your design.

## Resources

This section contains a list of resources to help you find design help. Your resource options include traditional Altera-based support such as online documentation, training, and My Support, as well as web-based forums and Wikis. The best option depends on your inquiry and your current stage in the design cycle.

## Support

Altera recommends that you seek support in the following order:

1. Search [www.altera.com](http://www.altera.com) for answers to your questions.  
Relevant literature appears on the [Altera literature pages](#), especially on the [Nios II Processor literature page](#) and the [SOPC Builder literature page](#).
2. Contact your local Altera sales office or sales representative, or your field application engineer (FAE).
3. Contact technical support at [www.altera.com/mysupport](http://www.altera.com/mysupport) to get support directly from Altera.
4. Consult the community-owned Nios Forum and Wiki:
  - [www.niosforum.com](http://www.niosforum.com)
  - [www.nioswiki.com](http://www.nioswiki.com)



Altera is not responsible for the contents of the Nios Forum and Nios Wiki websites, which are maintained by groups outside of Altera.

To learn how the tools work together and to use them in an instructor-led environment, register for training.

## Training

Several training options are available. For information about general training, refer to Altera's Education and Events website at [www.altera.com/education/edu-index.html](http://www.altera.com/education/edu-index.html).

For detailed information on available courses and their locations, visit the Altera Technical Training website at [www.altera.com/education/training/curriculum/embedded\\_sw/trn-embedded\\_sw.html](http://www.altera.com/education/training/curriculum/embedded_sw/trn-embedded_sw.html). This website contains information on both online and instructor-led training.

## Documentation

Documentation about the Nios II processor and embedded design is located in your Nios II EDS installation directory at `$SOPC_KIT_NIOS2/documents/index.htm`. To access this page directly on Windows platforms, on the Start menu, click **All Programs**. On the All Programs menu, on the Altera submenu, on the Nios II EDS `<version>` submenu, click **Nios II <version> Documentation**. This web page contains links to the latest Nios II documentation.

The Nios II literature page includes a list and links to available documentation at [www.altera.com/literature/lit-nio2.jsp](http://www.altera.com/literature/lit-nio2.jsp). At the bottom of this page, you can find links to various product pages that include Nios II processor online demonstrations and embedded design information.

Useful information for first time Nios II IDE users appears on the Welcome page. This page appears the first time you open the Nios II IDE after a new installation. You can also open it at any time from the Nios II IDE by clicking **Welcome** on the Help menu. A PDF version is also available at [www.altera.com/literature/ug/ug\\_nios2\\_ide\\_help.pdf](http://www.altera.com/literature/ug/ug_nios2_ide_help.pdf).

The other chapters in the *Embedded Design Handbook* are a valuable source of information about embedded hardware and software design, verification, and debugging. Each chapter contains links to the relevant overview documentation.

### Third Party Intellectual Property

Many third parties have participated in developing solutions for embedded designs with Altera FPGAs through the Altera AMPP<sup>SM</sup> Program. For up-to-date information on the third-party solutions available for the Nios II processor, refer to the Altera embedded processing web pages at [www.altera.com/embedded](http://www.altera.com/embedded), and click **Embedded Software Partners**.

Several community forums are also available. These forums are not controlled by Altera. The Nios Forum's Marketplace provides third-party hard and soft embedded systems-related IP. The forum also includes an unsupported projects repository of useful example designs. You are welcome to contribute to these forum pages.

Traditional support is available from the Support Center or through your local Field Application Engineer (FAE). You can obtain more informal support by visiting the Nios Forum at [www.niosforum.com](http://www.niosforum.com) or by browsing the information contained on the Nios Wiki, at [www.nioswiki.com](http://www.nioswiki.com). Many experienced developers, from Altera and elsewhere, contribute regularly to Wiki content and answer questions on the Nios Forum.

## Glossary

The following definitions explain some of the unique terminology for describing SOPC Builder and Nios II processor-based systems:

- **System interconnect fabric**—An interface through which the Nios II processor communicates to on- and off-chip peripherals. This fabric provides many convenience and performance-enhancing features.
- **Component**—A named module in SOPC Builder that contains the hardware and software necessary to access a corresponding hardware peripheral.
- **Custom instruction**—Custom hardware processing integrated into the Nios II processor's ALU. The programmable nature of the Nios II processor and SOPC Builder-based design supports this implementation of software algorithms in custom hardware. Custom instructions accelerate common operations. (The Nios II processor floating-point instructions are implemented as custom instructions).
- **Custom peripheral**—An accelerator implemented in hardware. Unlike custom instructions, custom peripherals are not connected to the CPU's ALU. They are accessed through the system interconnect fabric. (*See System interconnect fabric*). Custom peripherals off-load data transfer operations from the processor in data streaming applications.
- **ELF (Executable and Loadable Format)**—The executable format used by the Nios II processor. This format is arguably the most common of the available executable formats. It is used in most of today's popular Linux/BSD operating systems.

- **HAL (Hardware Abstraction Layer)**—A lightweight runtime environment that provides a simple device driver interface for programs to communicate with the underlying hardware. It provides a POSIX-like software layer and wrapper to the newlib C library.
- **Nios II C-To-Hardware Acceleration (C2H) Compiler**—A push-button ANSI C-to-hardware compiler that allows you to explore algorithm acceleration and design-space options in your embedded system.
- **Nios II Command Shell**—The command shell you use to access Nios II and SOPC Builder command-line utilities.
  - On Windows platforms, a Nios II command shell is a Cygwin bash with the environment properly configured to access command-line utilities.
  - On Linux platforms, to run a properly configured bash, type  
`$SOPC_KIT_NIOS2/sdk_shell` ←
- **Nios II Embedded Development Suite (EDS)**—The complete software environment required to build and debug software applications for the Nios II processor. The EDS includes the Nios II IDE. (*See Nios II IDE*).
- **Nios II IDE**—An Eclipse-based development environment for Nios II embedded designs that provides software project management, build, and debugging capabilities.
- **SOPC Builder**—Software that provides a GUI-based system builder and related build tools for the creation of FPGA-based subsystems, with or without a processor.

## Conclusion

This chapter is a basic overview of the Altera embedded development process and tools for the first time user. The chapter focuses on using these tools and where to find more information. It references other Altera documents that provide detailed information on the individual tools and procedures. It contains resource and glossary sections to help orient the first time user of Altera's embedded development tools for hardware and software development.

## Referenced Documents

This chapter references the following documents:

- *AN346: Using the Nios Development Board Configuration Controller Reference Designs*
- *AN428: MAX II CPLD Design Guidelines*
- *AN458: Alternative Nios II Boot Methods*
- *Archiving SOPC Builder Projects* chapter in volume 4 of the *Quartus II Handbook*
- *Debugging Nios II Designs* chapter of the *Embedded Design Handbook*
- *Developing Components for SOPC Builder* chapter in volume 4 of the *Quartus II Handbook*
- *Developing Nios II Software* chapter of the *Embedded Design Handbook*
- *Embedded Design Handbook*

- *Hardware Acceleration and Coprocessing* chapter of the *Embedded Design Handbook*
- *I/O Management* chapter in volume 2 of the *Quartus II Handbook*
- *Nios II Command-Line Tools* chapter of the *Embedded Design Handbook*
- *Nios II Custom Instruction User Guide*
- *Nios II Flash Programmer User Guide*
- *Nios II Hardware Development Tutorial*
- *Nios II Processor Reference Handbook*
- *Nios II Software Developer's Handbook*
- *Optimizing Nios II C2H Compiler Results* chapter of the *Embedded Design Handbook*
- *Remote Update Circuitry Megafunction User Guide (ALTREMOTE\_UPDATE)*
- *System Console User Guide*
- *Using the Nios II Software Build Tools* chapter of the *Nios II Software Developer's Handbook*
- *Verification and Board Bring-Up* chapter of the *Embedded Design Handbook*
- *Volume 4: SOPC Builder* of the *Quartus II Handbook*
- *Volume 5: Embedded Peripherals* of the *Quartus II Handbook*

## Document Revision History

Table 1–1 shows the revision history for this chapter.

**Table 1–1.** Document Revision History

<b>Date and Document Version</b>	<b>Changes Made</b>	<b>Summary of Changes</b>
January 2009 v2.1	Updated Nios Wiki hyperlink.	Updated Nios Wiki hyperlink.
November 2008 v2.0	Added System Console.	Added System Console.
March 2008 v1.0	Initial release.	—